Communication Avoiding Power Scaling

Power Scaling Derivatives of Algorithmic Communication Complexity

John D. Leidel, Yong Chen

Parallel Programming Models & Systems for High End Computing

(P2S2 2015)

Sept 1, 2015

TEXAS TECH

UNIVERSITY

DISCL

Overview

- Intro: Power limitations of scalable systems
- Energy Performance Scaling
- Algorithmic Techniques
- Algorithmic Experiments
- Energy Performance Scaling

Power limitations of scalable systems

INTRO

Power Limitations of Scalable Systems

- Current HPC systems are limited in scale due to hardware, software and power [facilities]
- Power has become a first order driver to scaling HPC platforms to the next major milestone
  - P. Kogge (editor). "Exascale Computing Study: Technology Challenges in Achieving Exascale," Univ. of Notre Dame, CSE Dept. Tech Report TR-2008-13, Sept. 28, 2008.
- Classic research on power has focused on:
  - *Power monitoring*: hardware and software techniques
  - *Power scaling*: largely reactive hardware and software techniques to meter power usage
- We present a tertiary area of research associated with classifying the power performance of scalable parallel algorithms

How scalable can my algorithm execute in terms of my facilities?

Governing equations behind determining energy performance efficiency

ENERGY PERFORMANCE SCALING

Energy Performance Equations

The governing equations for quantifying Energy Performance [EP] can be described as follows:

(1) $\boldsymbol{EP_p = EAvg_p / T_p}$; where ***EAvg*** = average peak power and ***T*** = runtime

(2) $\boldsymbol{EP_P = (EAvg_s + \max(EAvg_p)) / (T_s + \max(T_p))}$

where $\{T_s, EAvg_s\}$ = Sequentual code; $\{T_p, EAvg_p\}$ = Parallel code

(3) $\boldsymbol{EAvg_p} = \sum_{0}^{F} PPL'_p$

where PPL'_n is the Peak Power from one component power plane

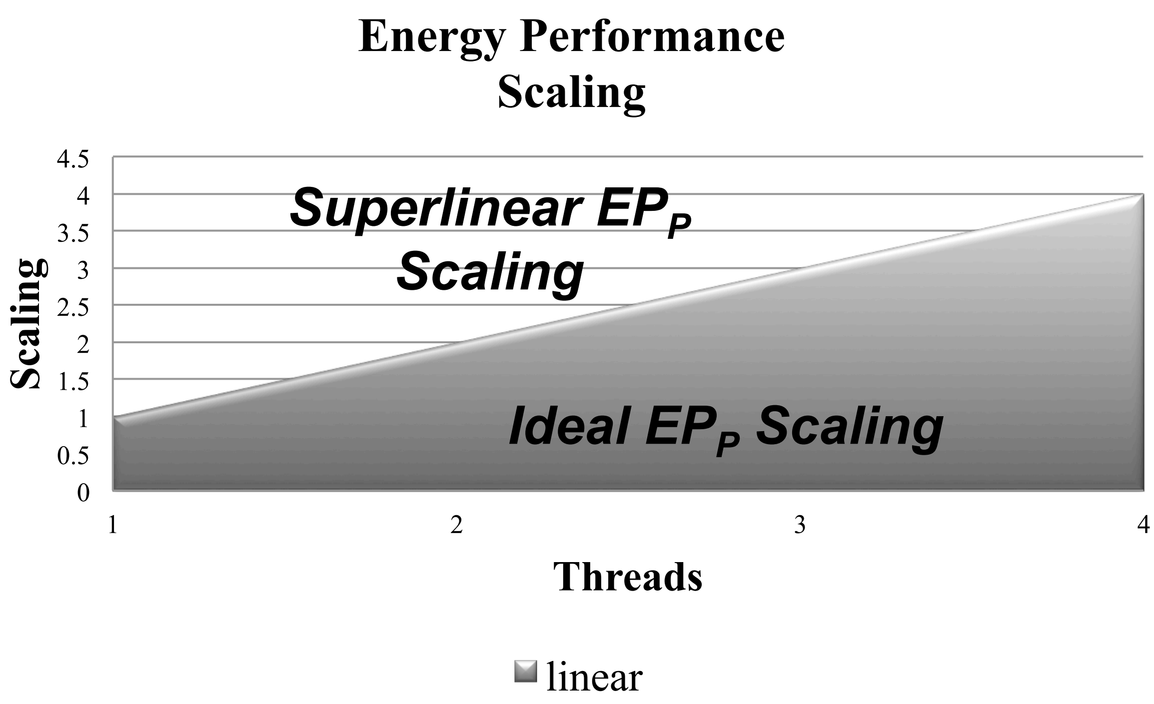
(3) $\boldsymbol{EP_P} = \left(\sum_{0}^{F} PPL'_s + \max\left(\sum_{0}^{F} PPL'_p\right)\right) / \left(T_s + \max(T_p) \right)$

(4) ***Scaling;*** $\boldsymbol{S(EP_P) = EP_P / EP_1}$

where $\boldsymbol{EP_P}$ = energy performance quantity for a given problem size using ***P*** parallel units

$\boldsymbol{EP_1}$ = energy performance quantity for a given problem size using 1 parallel unit

Energy Performance Scaling



- *Linear Scaling*
  - Best possible scenario where power and performance scaling are identical
- *Ideal Scaling*
  - Power scales at a rate less than performance scaling, or
  - Performance is significantly sub-linear
- *Superlinear Scaling*
  - Power scales at a rate greater than performance scaling

Matrix multiplication methodologies

ALGORITHMIC TECHNIQUES

Algorithmic Techniques

- We utilize classic double precision, square matrix multiplication as the basis for our research
  - BLAS: DGEMM
- We choose three algorithmic techniques:
  - ***OpenBLAS [CBLAS]***: Parallel Blocked (Tiled)
  - ***Classic Strassen-Winograd***: Recursive operation reduction
  - ***Communication Avoiding Parallel Strassen [CAPS]***: Two-stage recursive operation and communication reduction
- Known Issues?
  - Parallel Strassen techniques require sufficiently large problems in order to meet or exceed the performance of blocked techniques
  - Strassen has different numerical stability than blocked techniques

$$\frac{2(n/2)^3 \textit{ flop}}{y \textit{ Mflop/s}} = \frac{15 \text{ x } 32(n/2)^2 \textit{ Bytes}}{z \textit{ MB/s}}$$

or

$$n = \frac{480 \times y}{z}$$

DISCL

OpenBLAS: Blocked Matmul

- Classic method to partition matrices into bxb sub-blocks
  - Optimize the locality of the respective sub-blocks by prefetching into “fast” memory
- Excellent scaling on architectures with multi-level caches
  - Excellent performance characteristics even with large systems
  - Limited in performance to the theoretical peak of the system
    - *Still an N^3 algorithm*
- Very power hungry
  - Largest portions of the processor are frequently utilized: *cache*
- *OpenBLAS Implementation*
  - *Solver written in assembly*
  - *Utilizes SIMD units [AVX2]*
  - *Utilizes OpenMP worksharing*

```
//-- Block (Tiled) Matrix-Matrix
//-- C = A * B; where A,B,C are NxN matrices of bxb
//--    sub-blocks; where b=n/N
do i=1, N
    do j=1, N
        Read C(i,j) into L1 cache
        do k=1, N
            Read A(i,k) into L1 cache
            Read B(k,j) into L1 cache
            C(i,j) += A(i,k) * B(k,j)
        end do
        Write back C(i,j) to memory
    end do
end do
```

DISCL

Strassen-Winograd

- Recursive method to multiply square matrices
- Method:
  - Recursively partitions matrix and performs a series of 7 sub-matrix computations
  - Cutoff threshold triggers a switch to a dense solver [traditional n^3]
  - Possible to exceed theoretical peak performance
  - Requires sufficiently large problems
- Implementation based upon Barcelona OpenMP Task Suite Strassen
  - Utilizes OpenMP Tasks for parallelism across threads
  - Manually unrolls dense loops for good SIMD utilization
  - Cutoff threshold of N'=64

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} * \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

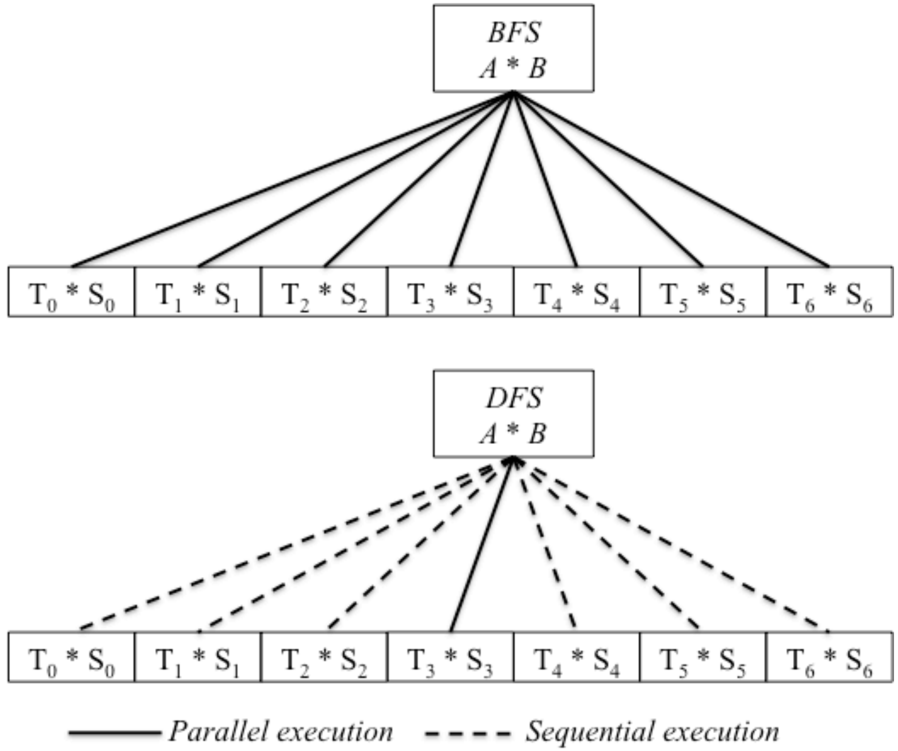
$$Q_1 = (A_{11} + A_{22}) * (B_{11} + B_{22})$$
$$Q_2 = (A_{21} + A_{22})\, B_{11}$$
$$Q_3 = A_{11} * (B_{12} - B_{22})$$
$$Q_4 = A_{22} * (B_{21} - B_{11})$$
$$Q_5 = (A_{11} + A_{12}) * B_{22}$$
$$Q_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$$
$$Q_7 = (A_{12} - A_{22}) * (B_{21} + B_{22})$$

$$C_{11} = Q_1 + Q_4 - Q_5 + Q_7$$
$$C_{12} = Q_3 + Q_5$$
$$C_{21} = Q_2 + Q_4$$
$$C_{22} = Q_1 - Q_2 + Q_3 + Q_6$$

Reducing operation count by trading multiplication for recursive addition

Communication Avoiding Parallel Strassen [CAPS]

- Derived from Strassen-Winograd and 2.5D techniques
  - Recursive implementation of Strassen
  - Represents matrix partitioning as a tree rather than tiles
- At each recursive depth, decide whether to use breadth-first or depth-first parallelism
  - ***BFS***: All 7 sub-problems executed in parallel [OpenMP Task]
  - ***DFS***: Each sub-problems executed sequentially, with parallelism [OpenMP Worksharing]




```
//-- CAPS Control Flow
if: DEPTH < CUTOFF_DEPTH
        Execute_Strassen_BFS
else:
        Execute_Strassen_DFS
```

We modify our Strassen implementation from BOTS and utilize a cutoff depth of 4

Test infrastructure, performance data and power data

ALGORITHMIC EXPERIMENTS

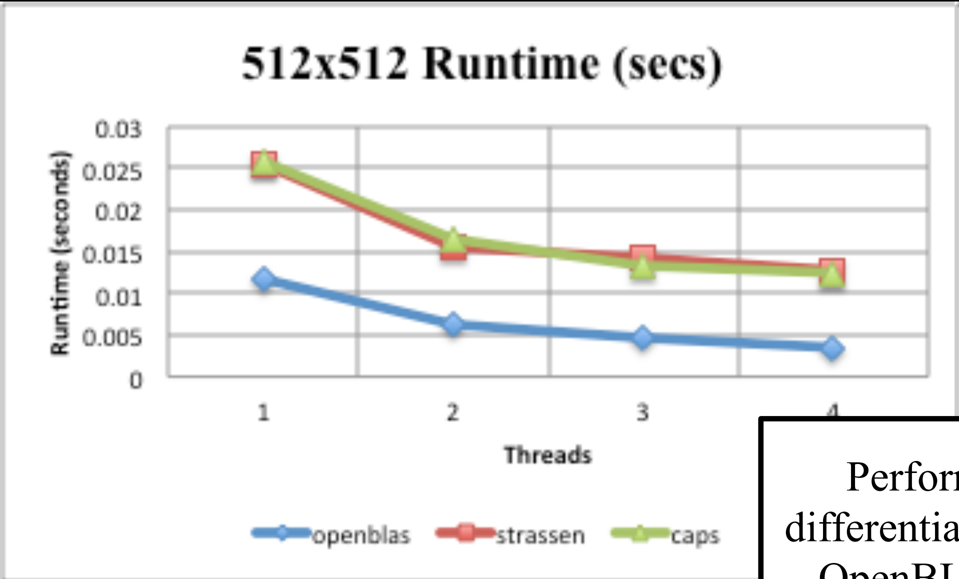
Test Platform

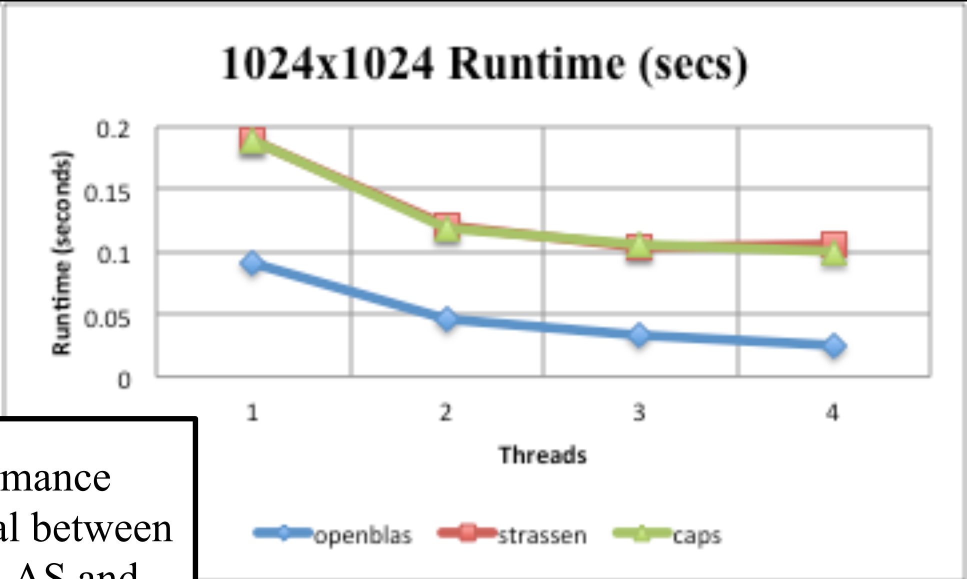
- Hardware
  - Lenovo TS140 server
  - Intel Xeon E3-1225 [Haswell]; Quad core 3.2Ghz; 8MB cache
  - DDR3-PC3-12800 DIMM w/ 4GB capacity
  - Power saving features disabled in BIOS
    - *Disables frequency scaling*
- Software
  - OpenSUSE 13.1; kernel: 3.11.10-7 x86_64
  - GNU GCC 4.8.1 20130909
    - *Use -march=avx2 where possible*
  - Barcelona OpenMP Task Suite 1.1.2 [modified]
  - OpenBLAS 0.2.8.0
  - PAPI 5.3.0
    - *Built with support for Intel RAPL:*
    - *http://icl.cs.utk.edu/projects/papi/wiki/PAPITopics:RAPL_Access*

Algorithmic Experiments

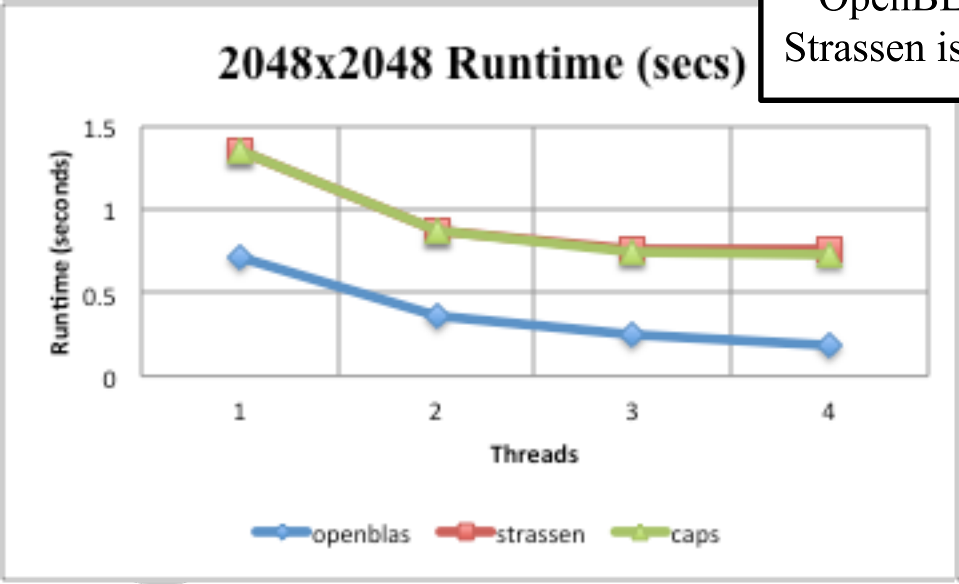
- Strassen_P Driver
  - Drives all tests using identical memory allocation
  - Initializes PAPI performance and power monitoring
  - Forces 60sec sleep period between tests
- Matrix Problem Sizes [*NxN*]
  - N = {512, 1024, 2048, 4096}
  - Larger problems are possible with OpenBLAS
  - Strassen requires additional buffer space
- Parallelism
  - Utilizes OpenMP thread counts = {1, 2, 3, 4}
  - OpenMP configured using OMP_NUM_THREADS environment variable
- Power Measurement
  - Power measured from within the driver using the PAPI RAPL component
  - Requires special permission to access system registers

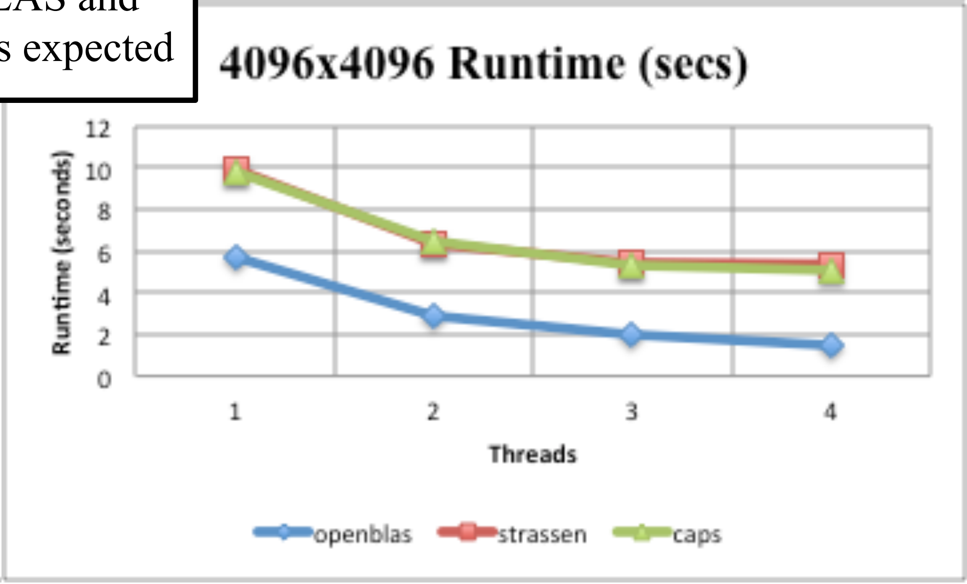
Performance



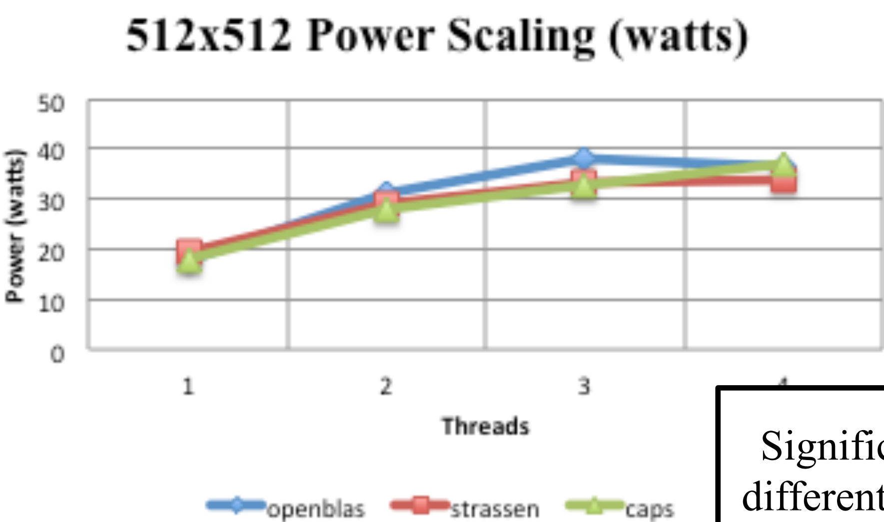


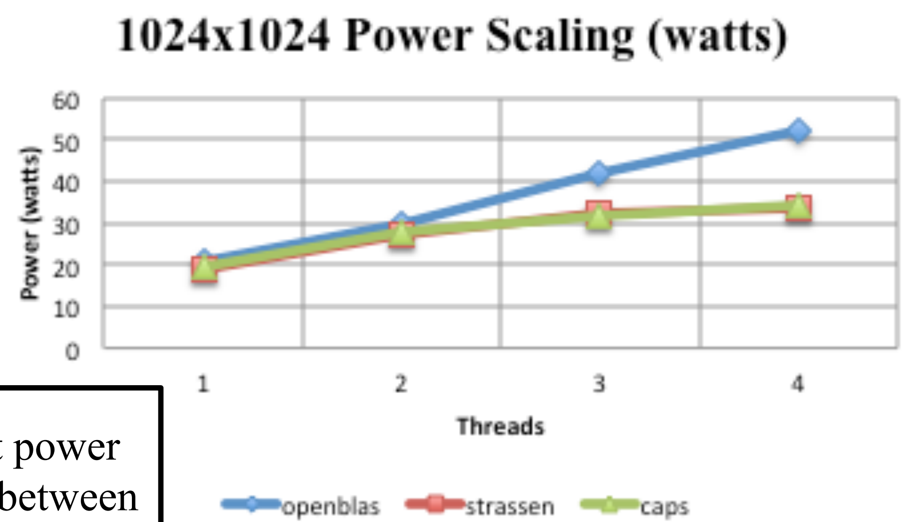
Performance differential between OpenBLAS and Strassen is expected



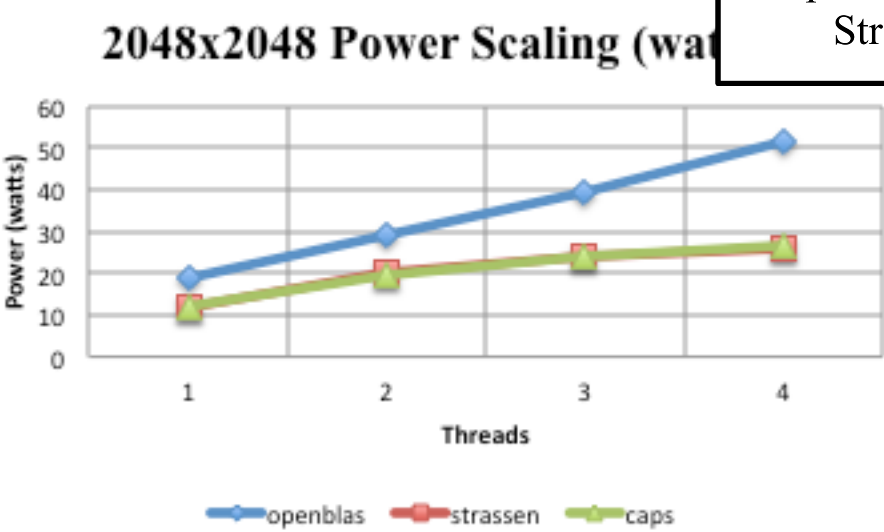


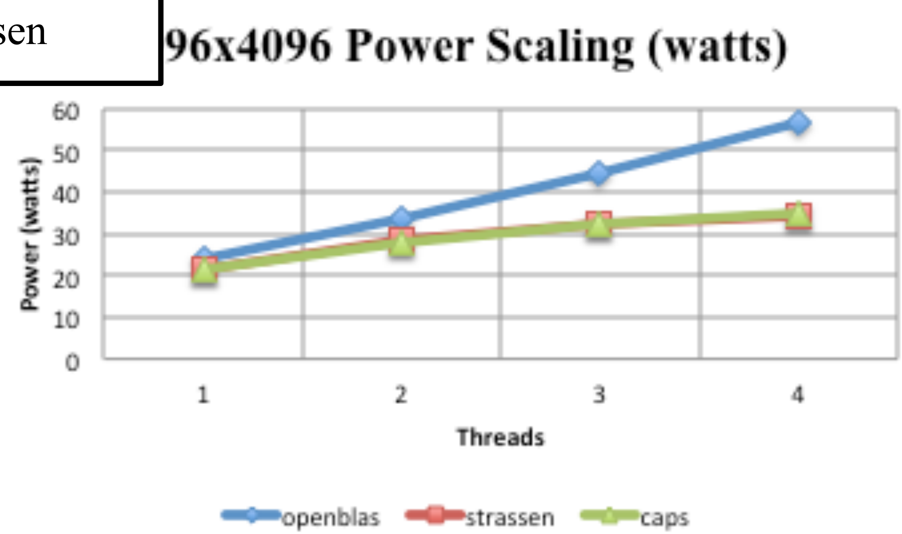
Power





Significant power differential between OpenBLAS and Strasssen

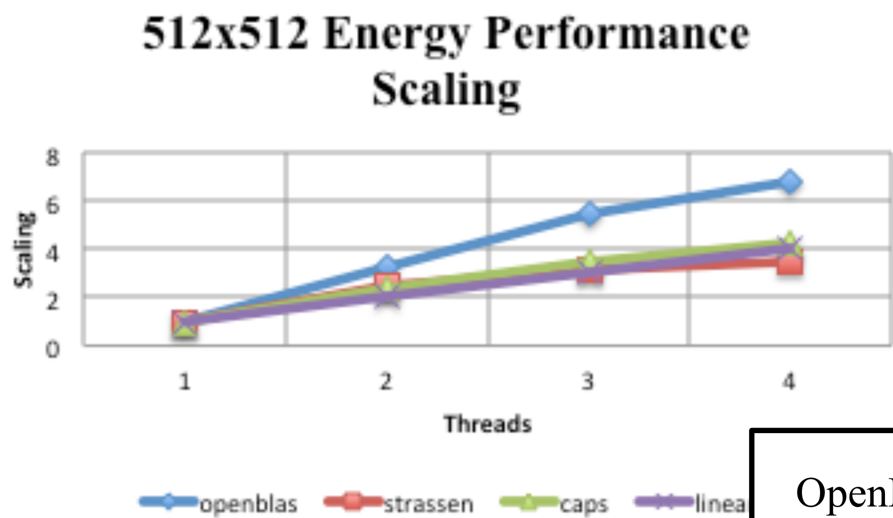


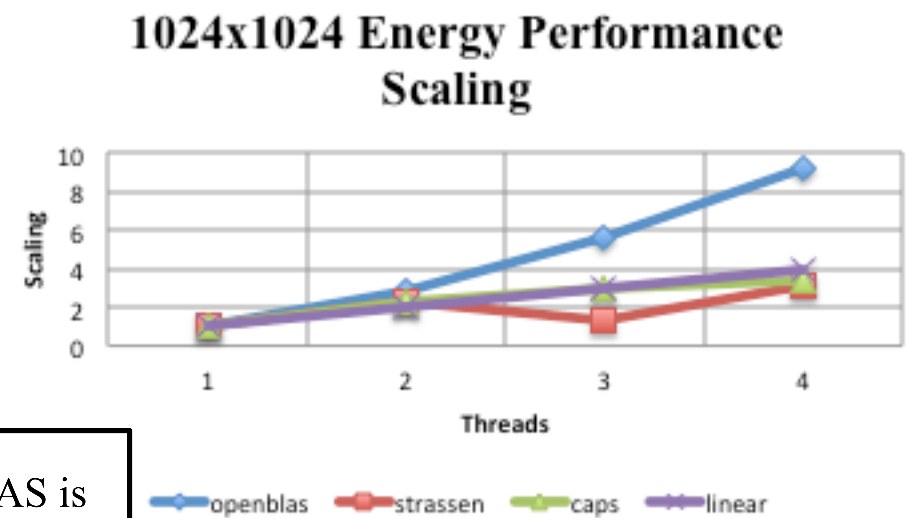


Utilizing our governing equations, examine our algorithmic efficiency

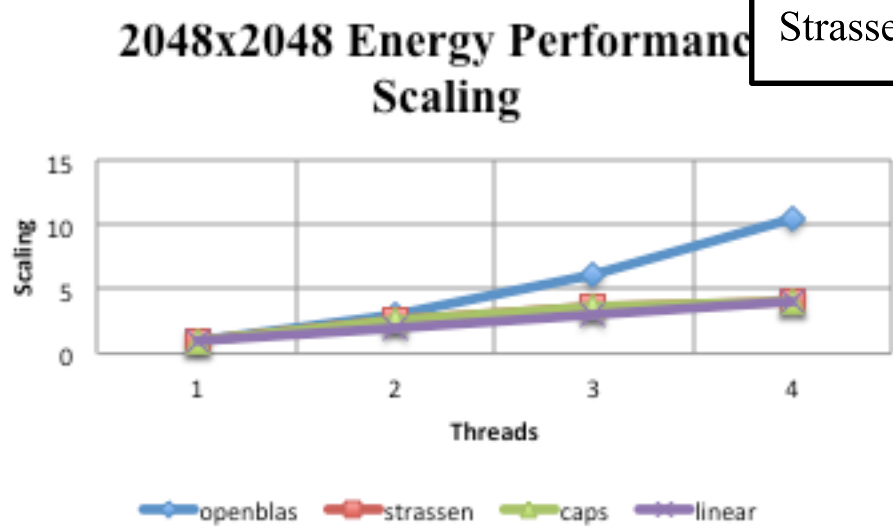
ENERGY PERFORMANCE SCALING

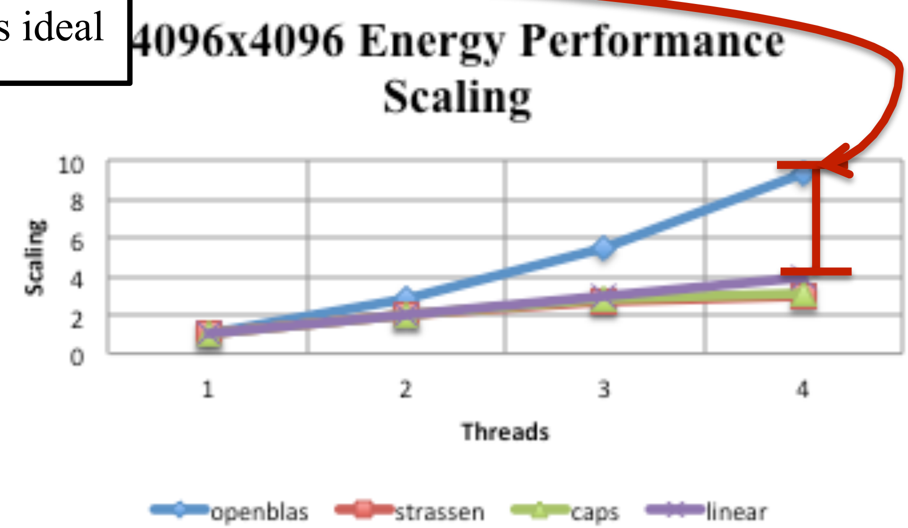
Energy Performance Scaling: S(EP_P)





OpenBLAS is **superlinear**
Strassen is ideal





Conclusions

- Governing equations to classify algorithmic complexity in terms of its energy performance efficiency: $\boldsymbol{EP_P}$
- Performance
  - OpenBLAS achieves highest performance on our SMP platform
  - CAPS is on average 5.97% faster than Strassen on our platform
- Power
  - OpenBLAS has the highest overall power
  - CAPS has an average power improvement of 2.59% over Strassen
- Energy Performance Scaling
  - OpenBLAS implementation is superlinear: power scales at a faster rate than performance
  - Strassen and CAPS fall within the ideal range
  - CAPS is slightly closer to the linear scale

Conclusion: CAPS provides the best EP_P scaling of all three approaches.

Future Work

- Additional Platform Measurement
  - Additional testing on more scalable Haswell systems
  - Measurement on forthcoming Skylake systems
  - How do these results vary on Xeon Phi or AMD APU systems?
- Additional Algorithm Measurement
  - Our aforementioned measurements were dense algorithms, what about sparse?
  - SPMV measurements using different storage techniques: CSR, CSC, raw, etc
- Power measurement Techniques
  - The component power measurement capabilities are still relatively limited
  - This is especially true on current/forthcoming memory devices (HBM, HMC)

TEXAS TECH UNIVERSITY™

References

G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. “Communications-optimal parallel algorithm for strassen’s matrix multiplication,” *CoRR*, abs/1202.3173, 2012.

G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. “Graph expansion and communication costs of fast matrix multiplication,” *J. ACM*, 59(6):32:1-32:23, Jan. 2013.

B. Lipshitz, G. Ballard, J. Demmel and O. Schwartz. “Communication-avoiding Parallel Strassen: Implementation and Performance,” Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC ’12, pp. 101:1-101:11, 2012.

K. Goto and R. Van De Geijn. “High-performance implementation of the level-3 blas”, ACM Trans. Math. Softw., 35(1) July 2008.

V. Weaver, M. Johnson, M. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore. “Measure Energy and Power with PAPI,” International Workshop on Power-Aware Systems and Architectures, Pittsburgh, PA, September 10, 2012.

Z. Xianyi, W Qian, Z, Yunquan, “Model-driven Level 3 BLAS Performance Optimization on Longsoon 3A Processor,” 2012 IEEE 18th International Conference on Parallel and Distributed Systems, 17-19 Dec.2012.

A. Duran, X. Teruel, R. Ferrer, X. Martorell, and E. Ayguade, “Barcelona OpenMP Tasks Suite: A Set of Benchmarks Targeting the Exploitation of Task Parallelism in OpenMP,” Proceedings of the 2009 International Conference on Parallel Processing (ICPP ’09). IEEE Computer Society, Washington, DC, USA, 124-131.

N.J. Higham, “Accuracy and Stability of Numerical Algorithms,” SIAM, Philadelphia, PA, 2nd edition, 2002.

References

OpenMP Architecture Review Board, “OpenMP Application Programming Interface Version 4.0.0,” July 2013.

P. Mucci, J. Dongarra, R. Kufrin, S. Moore, F. Song, and F. Wolf, “Automating the Large-Scale Collection and Analysis of Performance,” Proceedings of the 5th LCI International Conference on Linux Clusters: The HPC Revolution, Austin, Texas, May 18-20, 2004.

K.R. Wadleigh, I.L. Crawford, “Mathematical Kernels: The Building Blocks of High Performance,” in *Software Optimization for High Performance Computing*, 1st ed. Upper Saddle River, New Jersey: Prentice Hall PTR, 2000, ch. 10, sec. 10.9.1, pp 299-300.

V.V. Williams, “An Overview of the Recent Progress on Matrix Multiplication,” SIGAct News 43, 4, December 2012, 57-59.

IBM Systems, *IBM PowerExecutive 1.10 Installation and User’s Guide,* June 2006.

C. Lefurgy, X. Wang, and M. Ware. Power Capping: A Prelude to Power Shifting. *Cluster Computing*, 11, 2. June 2008, 183-195.

P. Bohrer et.al., The Case for Power Management in Web Servers. In R. Graybill and R. Melhem, editors, Power Aware Computing. Kluwer Academic Publishers, 2002.

H. Hoffman, S. Sidiroglou, M. Carbin, S. Misailovic, A. Argawal, and M. Rinard. Power-Aware Computing with Dynamic Knobs. Technical Report TR-2010-027, CSAIL, MIT, May 2010.

H. Hoffman, S. Sidiroglou, M. Carbin, A. Argawal, and M. Rinard. Dynamic Knobs for Response Power-Aware Computing. *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVI)*. ACM, New York, NY. 199-212.

References

Y.H. Lu et.al, "Operating-system Directed Power Reduction," *Int. Symp. on Low Power Electronics and Design*, 2000.

Q.Wu, et.al. "Fomral Control Techniques for Power-Performance management." IEEE Micro, 25(5), 52-62, 2005.

P. Kogge (editor). "Exascale Computing Study: Technology Challenges in Achieving Exascale," Univ. of Notre Dame, CSE Dept. Tech Report TR-2008-13, Sept. 28, 2008.

B.Grayson, A.Shah, R. van de Geijn. "A High Performance Strassen Implementation," Department of Computer Science, The University of Texas, TR-95-24, June 1995.

Q. Luo, J.B. Drake, "A Scalable Parallel Strassen's Matrix Multiplication Algorithm for Distributed Memory Computers," *Proceedings of the 1995 ACM Symposium on Applied Computing*. ACM, New York, NY. 221-226.

E. Solomonik and J. Demmel. "Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms". In Euro-Par'11:

Proceedings of the 17th International European Conference on Parallel and Distributed Computing. Springer, 2011.